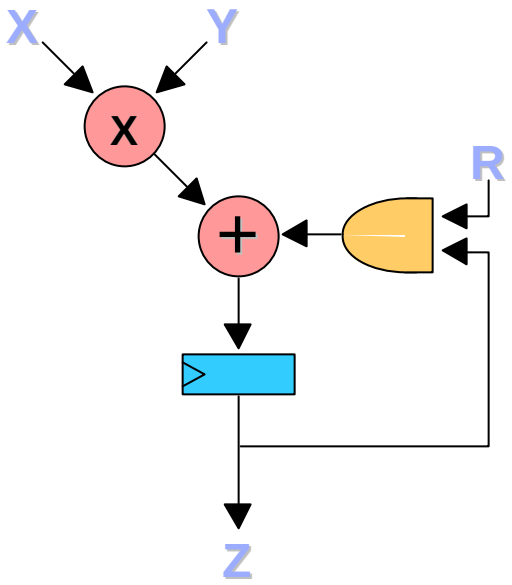
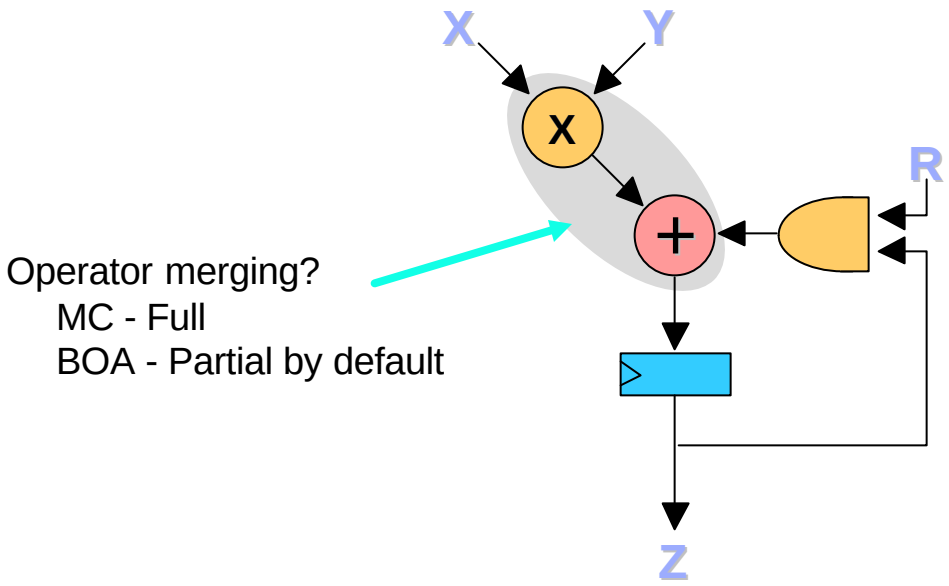
MACO - simple, binary




```
input signed [1] R;
input  [w] X,Y;
wire   [accw] XY = X*Y;
wire   [accw] ACCin = XY + (Z&R);
output  [accw] Z = sreg(ACCin);
```

MAC1 - simple, binary

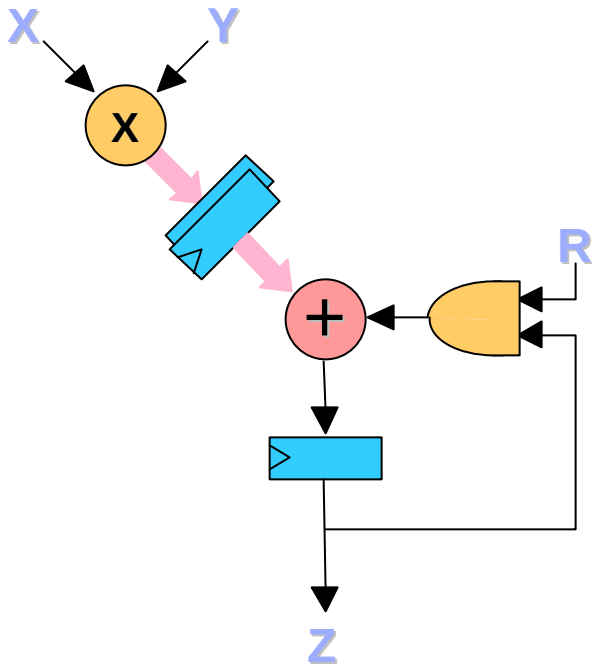


Operator merging?

- MC - Full
- BOA - Partial by default

```
input signed [1] R;
input  [w] X,Y;
wire   [accw] ACCin = X*Y + (Z&R);
output [accw] Z = sreg(ACCin);
```

MAC2 - carrysave multiplier




```
directive local (carrysave="convert");
wire   [accw] prod  = X*Y;
wire   [accw] ACCin = sreg(prod) + (Z&R);
output [accw] Z = sreg(ACCin);
```

Accessing Carrysave Signals Individually ?

- This is for experienced designers only
- csconvert() is undocumented because it is easy to get into trouble.
- It is only needed when you need to gate or mux signals in carrysave format

The csconvert() function is used as follows

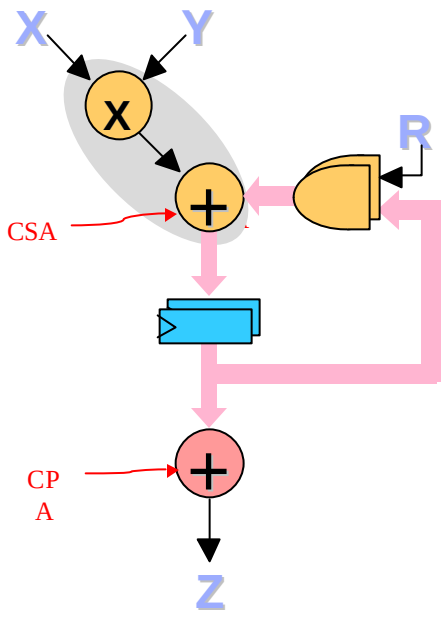
```
wire unsigned [3] X,Y;
wire unsigned [6] Z;
directive(carrysave=convert,multtype=nonbooth);
wire [1] prod = X*Y; // width&format irrelevant
wire unsigned [6] P0,P1; // width & format relevant
csconvert(P0,P1,prod);
directive(carrysave=off);
Z = P0+P1; // binary result
```

Modeling Restrictions of CS Terms P0 and P1

- **The individual bits of P0 and P1 can not be modeled at the RT Level of abstraction. They can only be modeled structurally.**
- **The arithmetic SUM of P0 and P1 can be modeled at the RT Level of abstraction.**
- **Consider 6*3 and 3*6**

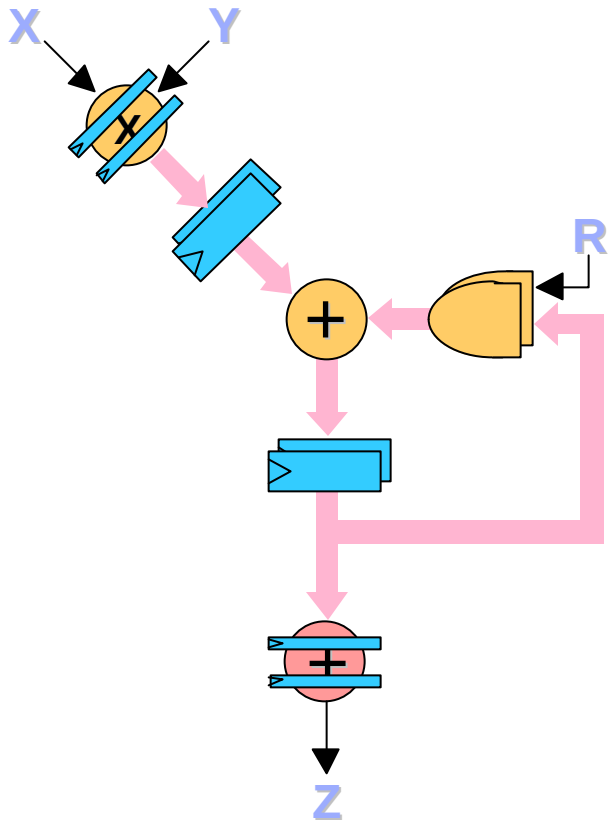
```
      110                    011
    x 011                  x 110
  -------                --------
      110                    000
     110.                   011.
+   000..              +   011..
  -------                --------
  001010   (P0)         001100   (P0)
+ 001000   (P1)       + 000110   (P1)
  -------                --------
  010010                 010010
```

MAC3 - carrysave accumulator




```
wire   [accw] ACC0,ACC1,ACCin0,ACCin1;
directive local (carrysave="convert");
wire   [accw] ACCin = X*Y + (ACC0&R) + (ACC1&R);
csconvert(ACCin0,ACCin1,ACCin);
ACC0 = sreg(ACCin0);
ACC1 = sreg(ACCin1);
output [accw] Z = ACC0+ACC1;
```

MAC4 - carrysave multiplier and accumulator




```
directive(fatype=fa, pipeline="on");
wire   [accw] ACC0,ACC1,ACCin0,ACCin1;
wire   [accw] prod,prodR;
directive local (carrysave="convert");
prod  = X*Y;
prodR = ResolveLatencyLoop(prod,3);
directive local(carrysave="convert",pipeline="off");
wire   [accw] ACCin = prodR + (ACC0&R) + (ACC1&R);
csconvert(ACCin0,ACCin1,ACCin);
ACC0 = sreg(ACCin0);
ACC1 = sreg(ACCin1);
directive local (pipeline="on");
wire [accw] Z_tmp = ACC0+ACC1;
output [accw] Z = ResolveLatency(Z_tmp,2);
```